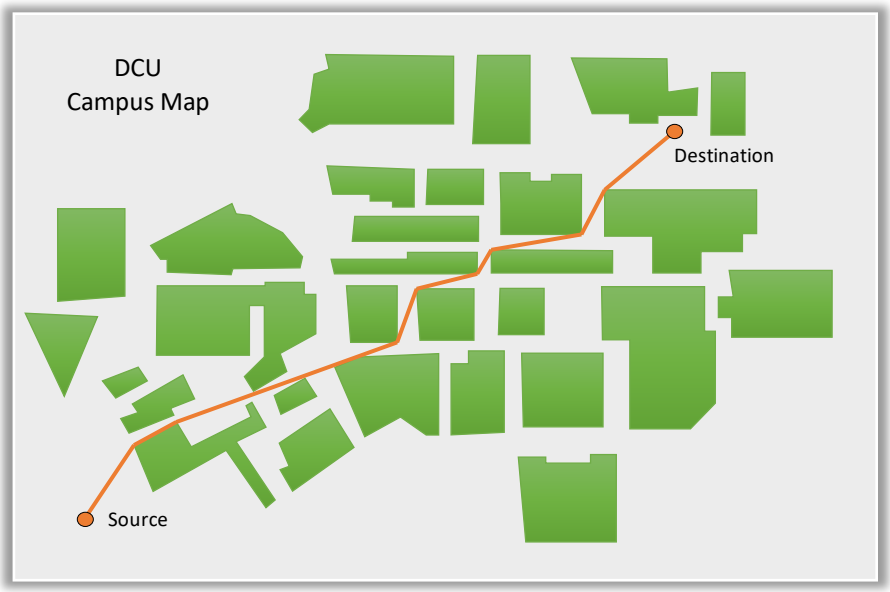
EE324 PROJECT

Convex Hulls and Path Finding

In this project you will design, implement and demonstrate an algorithm to find the shortest path, from a source to a destination point, avoiding a number of obstacles on a 2D plane. The obstacles have straight-line edges (simple polygons) but are not necessarily convex.



Example Problem Instance: A short path between two points on the DCU Campus

The first phase of your solution will be to find the convex hull of each obstacle shape using *Graham Scan*. The second phase of your algorithm will find the shortest path you can from source to destination, while avoiding the convex hulls of the shapes. The path finding algorithm you use is not specified. It is for you to choose/design an algorithm that achieves a good solution (not necessarily an optimal solution), which is:

- **Correct**: The path found is valid, avoiding all obstacles,
- **General**: Your algorithm will work correctly for any given set of shapes and is guaranteed to terminate with a solution,
- **Accurate**: Path length found is close to (or equal to) the shortest possible path,
- **Efficient**: Your algorithm has good computational complexity.

This document describes the task in detail and sets out the structure of the project report you are to turn in. You will also be required to demonstrate your solution in the Lab, after the project completion deadline.

This project contributes 25% to your overall EE324 grade.

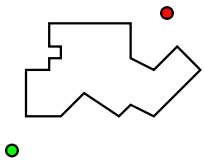
1 Algorithm Input

The input to your algorithm will be a file containing the points describing each obstacle. The points of each obstacle will be listed in order of occurrence travelling around the edge of the obstacle shape. An input file is a text file formatted as follows:

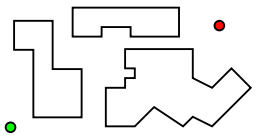
```
SRC-DST
Sx,Sy
Dx,Dy
POLYGON
p1X,p1Y
p2X,p2Y
. . .
POLYGON
p1X,p1Y
p2X,p2Y
. . .
END
```

`(Sx,Sy)` specifies the coordinates of the source point and `(Dx,Dy)` the coordinates of the destination. Keyword `POLYGON` indicates a new shape, which is followed by the coordinates of the shape's vertices. All `(X,Y)` coordinate values in the file will lie within the unit square, extending from `(0,0)` to `(1,1)`. Three test input files are provided to you for developing and testing your algorithms (depicted below). The green point indicates the starting point and the red point indicates the end point of the path to be found.

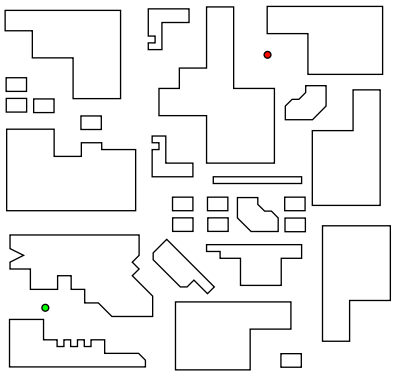
Your final Java programme will take an input filename as a command line parameter.



TEST-MAP-0.TXT



TEST-MAP-1.TXT



TEST-MAP-2.TXT

- Note that the given input polygons may have collinear points.
- There will be no overlap of the convex hulls of polygons in the given input data.
- The given source and destination points will lie *outside* of all convex hulls.

1.1 Input Data Handling

You are provided with the following classes to handle file input and to provide initial versions of classes you can use to start developing your code:

`ShapeMap, MapFileReader, Polygon2D, Point2D.`

ShapeMap stores a collection of polygons, with each polygon stored as an array of points. It has a constructor which makes a new ShapeMap object by loading polygon data (and source and destination points) from a specified input file, for example:

```
ShapeMap sm = new ShapeMap("TEST-MAP-1.TXT");
```

ShapeMap uses the MapFileReader class to handle input file processing. You will not need to call MapFileReader methods directly or modify this class.

Internally, ShapeMap stores a list of Polygon2D objects and allows retrieval of individual polygons from a ShapeMap using an *iterator*. That is, ShapeMap is defined as

```
public class ShapeMap implements Iterable<Polygon2D>
```

and to retrieve individual polygons from a ShapeMap the iterator can be used as follows:

```
for (Polygon2D poly : sm) {
      poly.draw();
}
```

Note that Polygon2D has a draw() method which draws the polygon on screen. The method uses Robert Sedgewick's StdDraw package, which you will use in this project to visualise your solution. An introduction to the package is given in Appendix 1 of this document.

Polygon2D objects are composed of Point2D objects. Each Point2D object stores the coordinates of a single polygon point (a vertex).

The points of a given polygon can be retrieved from a Polygon2D object using the method:

```
public Point2D[] asPointsArray()
```

Study the code of the ShapeMap, Polygon2D, Point2D classes. You may edit and extend them as you wish for use in your implementation. There is already enough code implemented in the classes to read a test map file and plot all the polygons read, using just a few lines of code as per above examples.

2 Convex Hull Implementation

You must use the Graham Scan algorithm, described in the course notes, to find the convex hull of each input polygon. It is suggested that you implement the algorithm within its own self-contained class and use the provided Polygon2D object to interface with the algorithm from other classes. For example, GrahamScan might be defined along these lines:

```
public class GrahamScan {
      . . .
      public static Polygon2D findConvexHull(Polygon2D polygon) { . . . }
      . . .
      private static Point2D lowestPoint(Point2D[] points) { . . . }
      private static void sortByAngleFrom(Point2D[] points, Point2D p) { . . . }
}
```

3 Required Algorithm Output

Your algorithm operation and output will be demonstrated graphically. The graphical output must be generated directly by your code (i.e. not sent to another programme for plotting). You may use only the `StdDraw` library. No other Java graphics package should be used. The following output is required:

(1) A drawing of all polygons, read from a given input file, with the convex hull of each polygon overlaid on this map. Use two different colours to make the convex hulls clearly distinguishable from the input polygons. Once your path finding algorithm has completed, the above drawing is to be overlaid with the solution to the path joining the given source and destination points. Use a third colour to clearly distinguish the path.

(2) Once the above drawing has completed, your program will print to standard output:

- The total execution time in milliseconds of all code
- The length of the path that was found

The total execution time will be measured by the difference in time between execution of the last and the first statement in your `main()` function, i.e. using the following code:

```
public static void main(String args[]) {
        final long startTime = System.currentTimeMillis();

        // all other code in main here . . .

        final long runTime = System.currentTimeMillis() - startTime;
        System.out.println("Execution time (ms)  = " + runTime);
        System.out.println("Length of path found = " + pathLength);
}
```

(3) In addition to above, your program will take a command line option –V. If –V is present on the command line, your graphical output will include additional drawing animation to illustrate what options/decisions your path-finding algorithm is considering as it executes. It is open as to how you choose to do this and will depend on your chosen path-finding algorithm. *You do not need to animate the operation of Graham Scan.*

4 Project Report Structure and Marking

You will submit a report *structured according to the headings below* and also include all of your Java code as an appendix to your report. Adhere to the maximum number of pages indicated for each section. The report contributes **90%** of the project marks. The demonstration of your solution in the Lab (after report submission) contributes **10%** of the marks. *This is an individual project. University plagiarism policy will apply in full. Your project report will be submitted via Loop and will be analysed using Turnitin text similarity analysis software.*

1 Description of Algorithm [6 Pages Max] [60 Marks]

Give a detailed description and explanation of the algorithm you used to find the shortest path. Your description should make use of appropriate diagrams and pseudo-code to give a full and precise description of your algorithm. Include screen-shots of your algorithm output and also execution time and length of path found. Include full descriptions of any sub-routines used by the algorithm (e.g. a sub-routine to determine if a point is inside a convex hull – depending on your solution approach).

This section will be marked based on how well it addresses the follow questions:

Is the **description of the algorithm clear and complete**? E.g. would it be possible for another person to fully understand and implement your algorithm given your description? [20 marks]

Correctness of algorithm. Is it clear from the description that the algorithm is correct? i.e. will it always terminate with a solution and will handle any valid input map data? [20 marks]

Accuracy and Efficiency of algorithm. How close to an optimal solution would the algorithm be expected to achieve? Is the algorithm expected to be efficient? [20 marks]

2 Implementation Design [3 Pages Max] [20 Marks]

Give a brief overview of how you extended the classes/data-structures provided and added your own classes/data-structures to complete your implementation. Describe the functionality provided by each class. Use an appropriate diagram to indicate the relationship between the classes. [10 Marks]

For your Graham Scan implementation, give a brief overview of how you implemented the algorithm. How were collinear points in an input polygon handled? What method was used to sort points by angle and to find turning directions? Did you use a pre-processing step on the input points to reduce the problem size? [10 Marks]

3 Algorithm Analysis [2 Pages Max] [10 Marks]

Perform an analysis of the computational complexity of your path-finding algorithm, in terms of the input problem size. Depending on the algorithm used, the input size will be determined by the number of input polygons and/or the total number of vertex points. Include any other variables that determine computational complexity of your particular algorithm.

4 Code Appendix [Number of Pages as Required]

Include all of your program code. If you have modified any of the given classes, also include the modified code.

Total marks for report [90 Marks]

Appendix 1 - StdDraw

StdDraw (part of the StdLib libaray) is a Java library useful for creating simple visualisations and animations of algorithms. It is written by Robert Sedgewick and Kevin Wayne of the Department of Computer Science at Princeton University.

To use the library you will need to download the stdlib-package.jar file, available from: http://introcs.cs.princeton.edu/java/stdlib/. The library is well documented and is very easy to use, as illustrated in the simple example programme below.

```
//
// StdDraw Example
//
package drawTest;
//
// StdDraw needs the stdlib-package.jar
// Download it from http://introcs.cs.princeton.edu/java/stdlib/
// and put the file somewhere in your project directory.
// Then add the jar file to the libraries for your project in Eclipse:
// Right-click on your project in Package Explorer, then select Build Path >
// Configure Build Path..
// Under Libraries tab click Add JARs.. and select your stdlib-package.jar file.
//
import edu.princeton.cs.introcs.StdDraw;
import java.awt.Color; // needed to use predefined colors in your drawings

public class DrawTest {
	public static void main(String args[]) {
		StdDraw.setCanvasSize(600,600);
		StdDraw.setPenColor(250, 255, 100);
		StdDraw.filledCircle(0.5, 0.5, 0.4);
		StdDraw.setPenColor(Color.BLACK);
		StdDraw.setPenRadius(0.01);
		StdDraw.circle(0.5, 0.5, 0.4);
		StdDraw.line(0.2, 0.4, 0.15, 0.45);
		StdDraw.line(0.2, 0.4, 0.8, 0.4);
		StdDraw.line(0.85, 0.45, 0.8, 0.4);
		StdDraw.filledCircle(0.35, 0.7, 0.05);
		StdDraw.filledCircle(0.65, 0.7, 0.05);
		StdDraw.pause(1000);
		StdDraw.setPenColor(250, 255, 100);
		StdDraw.pause(1000);
		StdDraw.filledCircle(0.35, 0.7, 0.06);
		StdDraw.setPenColor(Color.BLACK);
		StdDraw.line(0.31,0.7,0.39,0.7);
		StdDraw.pause(1000);
		StdDraw.filledCircle(0.35, 0.7, 0.05);
	}
}
```